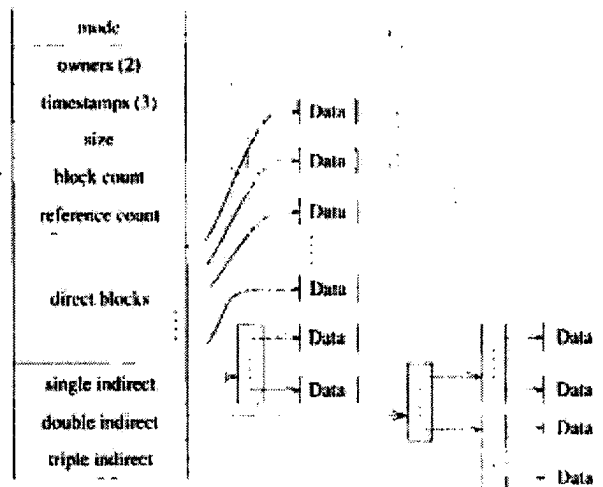


# Exhibit G-2

2. A method as in claim 1, wherein said file system on said storage system always moves atomically from said first consistent state to said second consistent state.	Leffler teaches that the file system advances during the periodic <i>sync</i> process when dirty buffers, including root inode, are forced to disk. <i>See</i> Leffler Sec. 7.4, pp.206-07; <i>see also e.g.</i> Bach, Sec. 4.1.3, p. 67.
3. A method as in claim 2, wherein said file system on said storage system moves atomically to said second consistent state by flushing said changes from said buffers to said storage system before updating said on-disk root inode with information from said incore root inode.	Leffler teaches that the file system advances during the periodic <i>sync</i> process when dirty buffers, including root inode, are forced to disk. <i>See</i> Leffler Sec. 7.4, pp.206-07; <i>see also e.g.</i> Bach, Sec. 4.1.3, p. 67.
4. A method as in claim 3, wherein updating said on-disk root inode further comprises updating said on-disk root inode and then a copy of said on-disk root inode such that if updating said on-disk root inode is interrupted, said copy of said on-disk root inode still points to said first consistent state of said file system.	Replicating the metadata, such as a root inode, is a technique that is well known in the art. Leffler teaches that in the Unix filesystem the superblock is replicated to allow recovery from crashes that corrupt the primary copy of the superblock. Leffler, Sec. 7.3, p. 196. In view of the knowledge in the art and Leffler's teaching, it would have been obvious for one of ordinary skill to make a copy of the root inode to recover from crashes that corrupt the primary copy of the root inode.
9. A device comprising: a processor; a memory; and	The file system taught in Leffler is inherently executed on a processor with a memory.
a storage system including one or more hard disks;	Leffler teaches a file system using a cache memory and one or more disks.
wherein said memory and said storage system store a file system; and	The file system in Leffler is stored in memory and on a storage system.
wherein said memory also stores information including instructions executable by said processor to maintain said file system, the instructions including steps of (a) maintaining an on-disk root inode on said storage system, said on-disk root inode pointing directly and indirectly to a first set of blocks on said storage system that store a first consistent state of said file system, and (b) maintaining an incore root inode in said memory, said incore root inode pointing directly and indirectly to	Leffler teaches maintaining an on-disk root inode pointing directly and indirectly to a set of blocks on disk in a consistent state:

buffers in said memory and a second set of blocks on said storage system, said buffers and said second set of blocks storing data and metadata for a second consistent state of said file system, said second set of blocks including at least some blocks in said first set of blocks, with changes between said first consistent state and said second consistent state being stored in said buffers and in ones of said second set of blocks not pointed to by said on-disk inode.

**Figure 7.6** The structure of an inode.



See Leffler Sec. 7.2

Leffler further teaches that a copy of the root inode (inode describing the root directory) and copies of lower level inodes and data blocks are maintained in memory (in-core). Leffler, Sec. 7.4, pp. 203-07. When the system writes new data, it allocates buffers in memory for the new data. See e.g. Leffler, Sec. 7.5, pp. 211-12. Periodically, the buffers are flushed to disk during a *sync* process. See e.g. Leffler, Sec. 7.4, p. 207. It is inherent that the incore root inode is updated to point to the new buffers and blocks. For instance, Bach (a conventional well known Unix textbook) teaches that in Unix, the in-core inode is updated when data blocks change. Bach, Sec. 4.1, pp. 62-63.

10. A device as in claim 9, wherein said file system on said storage system always moves atomically from said first consistent state to said second consistent state.

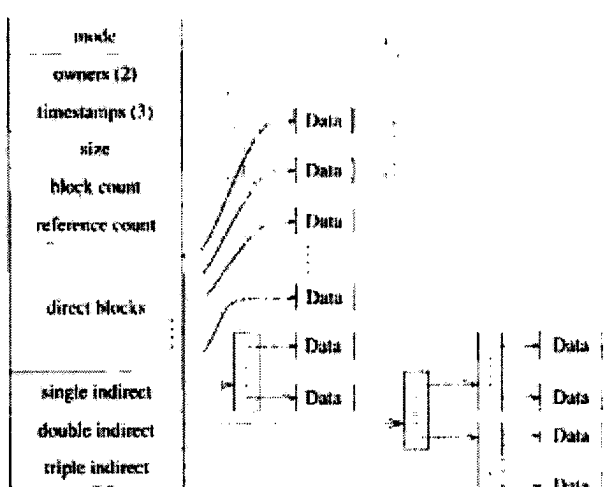
Leffler teaches that the file system advances during the periodic *sync* process when dirty buffers, including root inode, are forced to disk. See Leffler Sec. 7.4, pp.206-07; see also e.g. Bach, Sec. 4.1.3, p. 67.

11. A device as in claim 10, wherein said file system on said storage system moves atomically to said second consistent state by flushing said changes from said buffers to said storage system before updating said on-disk root inode with information from said incore root inode.

Leffler teaches that the file system advances during the periodic *sync* process when dirty buffers, including root inode, are forced to disk. See Leffler Sec. 7.4, pp.206-07; see also e.g. Bach, Sec. 4.1.3, p. 67.

12. A device as in claim 11, wherein

Replicating the metadata, such as a root inode, is a

<p>updating said on-disk root inode further comprises updating said on-disk root inode and then a copy of said on-disk root inode such that if updating said on-disk root inode is interrupted, said copy of said on-disk root inode still points to said first consistent state of said file system.</p>	<p>technique that is well known in the art. Leffler teaches that in the Unix filesystem the superblock is replicated to allow recovery from crashes that corrupt the primary copy of the superblock. Leffler, Sec. 7.3, p. 196. In view of the knowledge in the art and Leffler's teaching, it would have been obvious for one of ordinary skill to make a copy of the root inode to recover from crashes that corrupt the primary copy of the root inode.</p>
<p>17. An article of manufacture comprising a machine-readable storage medium storing instructions executable by a processor coupled to a memory and to a storage system, said storage system comprising one or more hard disks, said memory and said storage system storing a file system, wherein the instructions, when executed by the processor, cause the processor to (a) maintain an on-disk root inode on said storage system, said on-disk root inode pointing directly and indirectly to a first set of blocks on said storage system that store a first consistent state of said file system, and (b) maintain an incore root inode in said memory, said incore root inode pointing directly and indirectly to buffers in said memory and a second set of blocks on said storage system, said buffers and said second set of blocks storing data and metadata for a second consistent state of said file system, said second set of blocks including at least some blocks in said first set of blocks, with changes between said first consistent state and said second consistent state being stored in said buffers and in ones of said second set of blocks not pointed to by said on-disk inode.</p>	<p>The file system in Leffler is inherently stored on machine readable medium in the form of instructions executable by a processor coupled to a memory and one or more hard disks.</p> <p>Leffler teaches maintaining an on-disk root inode pointing directly and indirectly to a set of blocks on disk in a consistent state:</p> <p><b>Figure 7.6</b> The structure of an inode.</p>  <p>See Leffler Sec. 7.2</p> <p>Leffler further teaches that a copy of the root inode (inode describing the root directory) and copies of lower level inodes and data blocks are maintained in memory (in-core). Leffler, Sec. 7.4, pp. 203-07. When the system writes new data, it allocates buffers in memory for the new data. See e.g. Leffler, Sec. 7.5, pp. 211-12. Periodically, the buffers are flushed to disk during a <i>sync</i> process. See e.g. Leffler, Sec. 7.4, p. 207. It is inherent that the incore root inode is updated to point to the new buffers and blocks. For instance, Bach (a conventional well known Unix textbook) teaches that in Unix, the in-core inode is updated when data blocks change. Bach, Sec. 4.1, pp. 62-63.</p>

18. An article of manufacture as in claim 17, wherein the instructions further cause the processor to move atomically said file system on said storage system from said first consistent state to said second consistent state.	Leffler teaches that the file system advances during the periodic <i>sync</i> process when dirty buffers, including root inode, are forced to disk. <i>See</i> Leffler Sec. 7.4, pp.206-07; <i>see also e.g.</i> Bach, Sec. 4.1.3, p. 67.
19. An article of manufacture as in claim 18, wherein the instructions cause the processor to move atomically said file system on said storage system to said second consistent state by flushing said changes from said buffers to said storage system before updating said on-disk root inode with information from said incore root inode.	Leffler teaches that the file system advances during the periodic <i>sync</i> process when dirty buffers, including root inode, are forced to disk. <i>See</i> Leffler Sec. 7.4, pp.206-07; <i>see also e.g.</i> Bach, Sec. 4.1.3, p. 67.
20. An article of manufacture as in claim 19, wherein updating said on-disk root inode comprises updating said on-disk root inode and then a copy of said on-disk root inode such that if updating said on-disk root inode is interrupted, said copy of said on-disk root inode still points to said first consistent state of said file system.	Replicating the metadata, such as a root inode, is a technique that is well known in the art. Leffler teaches that in the Unix filesystem the superblock is replicated to allow recovery from crashes that corrupt the primary copy of the superblock. Leffler, Sec. 7.3, p. 196. In view of the knowledge in the art and Leffler's teaching, it would have been obvious for one of ordinary skill to make a copy of the root inode to recover from crashes that corrupt the primary copy of the root inode.

#### Sixth Basis of Invalidity


The references applicable to the sixth basis of invalidity are:


1. Rosenblum, Ousterhoot, *The LFS Storage Manager*, presented at USENIX Tech. Conf., Anaheim, CA, 1990 ("Rosenblum").
2. Ylonen, as cited hereinabove.
3. Leffler, as cited hereinabove.

The pertinence and manner of applying Rosenblum, Ylonen and Leffler claims 1-24 for which re-examination is requested is as follows:


Claims of '211 Patent	Rosenblum, Ylonen and Leffler
-----------------------	-------------------------------


1. A method of maintaining a file system stored in a memory and on a storage system that includes one or more hard disks, said method comprising steps of:	Rosenblum teaches a file system stored in memory and on hard disks.
maintaining an on-disk root inode on said storage system, said on-disk root inode pointing directly and indirectly to a first set of blocks on said storage system that store a first consistent state of said file system; and	Rosenblum maintains an on disk root inode. The format of the inodes and indirect blocks is the same as in the standard Unix file system, which is known to one of ordinary skill to provide for root inodes that directly and indirectly point to data blocks. <i>See</i> Rosenblum, Sec. 4.2.1, p.6.
maintaining an incore root inode in said memory, said incore root inode pointing directly and indirectly to buffers in said memory and a second set of blocks on said storage system, said buffers and said second set of blocks storing data and metadata for a second consistent state of said file system, said second set of blocks including at least some blocks in said first set of blocks, with changes between said first consistent state and said second consistent state being stored in said buffers and in ones of said second set of blocks not pointed to by said on-disk inode.	Rosenblum teaches that file system metadata, including inodes, and data blocks are cached in memory, where changes are accumulated before being forced to disk. Rosenblum, Sec. 4.1, p. 5. Before being forced to disk, the modified in-core root inode inherently points to buffers in memory containing modified data, new blocks already forced to disk, and old blocks with unmodified data.
2. A method as in claim 1, wherein said file system on said storage system always moves atomically from said first consistent state to said second consistent state.	Rosenblum teaches a checkpoint, which marks a consistent state of the file system. Rosenblum, Sec. 4.4.1, p. 9. Once all modifications are written to disk, a checkpoint region is written, atomically moving the file system to a new consistent state. <i>Id.</i> The checkpoint region points to the last segment written and the location of the inode map. <i>Id.</i>
3. A method as in claim 2, wherein said file system on said storage system moves atomically to said second consistent state by flushing said changes from said buffers to said storage system before updating said on-disk root inode with information from said incore root inode.	Rosenblum teaches a checkpoint, which marks a consistent state of the file system. Rosenblum, Sec. 4.4.1, p. 9. Once all modifications are written to disk, a checkpoint region is written, atomically moving the file system to a new consistent state. <i>Id.</i> The checkpoint region points to the last segment written and the location of the inode map. <i>Id.</i>
4. A method as in claim 3, wherein	Replicating the metadata, such as a root inode, is a

updating said on-disk root inode further comprises updating said on-disk root inode and then a copy of said on-disk root inode such that if updating said on-disk root inode is interrupted, said copy of said on-disk root inode still points to said first consistent state of said file system.	technique that is well known in the art. For instance, Leffler teaches that in the Unix filesystem the superblock is replicated to allow recovery from crashes that corrupt the primary copy of the superblock. Leffler, Sec. 7.3, p. 196. In view of the knowledge in the art and, it would have been obvious for one of ordinary skill to make a copy of the root inode to recover from crashes that corrupt the primary copy of the root inode.
5. A method as in claim 1, further comprising the step of creating a snapshot of said file system by copying only said on-disk root inode.	Rosenblum teaches that the group commit concept at the basis of atomic updating taught in Rosenblum is found database literature. Rosenblum, Sec. 4.5, p. 10. Therefore, one of ordinary skill in the art would find motivation to look to the teachings of database art, such as Ylonen. Ylonen teaches an atomic commit method for databases, wherein a page table pointer (equivalent to a root inode) is updated to atomically advance the database to the next consistent state. Ylonen, Sec. 2, pp. 1-2, Fig. 1. Ylonen further teaches taking one or more snapshots by saving the address of the page table and preventing freeing of data pages. Ylonen, Sec. 8, p. 4, Fig. 2. The page table address is in the page table pointer. Thus, every time the page table pointer is copied (an operation that occurs at every commit cycle), "[w]e can have an arbitrary number of snapshots at different times, and they will all be consistent as long as their pages are not freed." <i>Id.</i>
6. A method as in claim 5, wherein when said snapshot is created, said snapshot and said file system share said first set of blocks on said storage system.	<p>Ylonen teaches a shadow paging (copy-on-write) technique that "never modifies accessible pages," therefore a snapshot and the active file system share unmodified blocks. Ylonen, Sec. 8, p. 4, Figs. 1 and 2.</p>  <p>Figure 2: Snapshots represent consistent database states as of some time in the past.</p>
7. A method as in claim 1, further comprising the step of creating plural snapshots of said file system by copying only said on-disk root inode at different times.	Ylonen teaches an atomic commit method for databases, wherein a page table pointer (equivalent to a root inode) is updated to atomically advance the database to the next consistent state. Ylonen, Sec. 2, pp. 1-2, Fig. 1. Ylonen further teaches taking one or



	more snapshots by saving the address of the page table and preventing freeing of data pages. Ylonen, Sec. 8, p. 4, Fig. 2. The page table address is in the page table pointer. Thus, every time the page table pointer is copied (an operation that occurs at every commit cycle), “[w]e can have an arbitrary number of snapshots at different times, and they will all be consistent as long as their pages are not freed.” <i>Id.</i>
8. A method as in claim 7, wherein when each one of said plural snapshots is created, each one of said snapshots and said file system share said first set of blocks on said storage system.	<p>Ylonen teaches a shadow paging (copy-on-write) technique that “never modifies accessible pages,” therefore a snapshot and the active file system share unmodified blocks. Ylonen, Sec. 8, p. 4, Figs. 1 and 2.</p>  <p>Figure 2: Snapshots represent consistent database states as of some time in the past.</p>
9. A device comprising: a processor; a memory; and	The file system taught in Rosenblum is inherently executed on a processor with a memory.
a storage system including one or more hard disks;	Rosenblum teaches a file system using a cache memory and one or more disks.
wherein said memory and said storage system store a file system; and	The file system in Rosenblum is stored in memory and on a storage system.
wherein said memory also stores information including instructions executable by said processor to maintain said file system, the instructions including steps of (a) maintaining an on-disk root inode on said storage system, said on-disk root inode pointing directly and indirectly to a first set of blocks on said storage system that store a first consistent state of said file system, and (b) maintaining an incore root inode in said memory, said incore root inode pointing directly and indirectly to buffers in said memory and a second set of blocks on said storage system, said buffers and said second set of blocks storing data and metadata for a second consistent state of said file	<p>Rosenblum maintains an on disk root inode. The format of the inodes and indirect blocks is the same as in the standard Unix file system, which is known to one of ordinary skill to provide for root inodes that directly and indirectly point to data blocks. <i>See Rosenblum, Sec. 4.2.1, p.6.</i></p> <p>Rosenblum further teaches that file system metadata, including inodes, and data blocks are cached in memory, where changes are accumulated before being forced to disk. Rosenblum, Sec. 4.1, p. 5. Before being forced to disk, the modified in-core root inode inherently points to buffers in memory containing modified data, new blocks already forced to disk, and old blocks with unmodified data.</p>

system, said second set of blocks including at least some blocks in said first set of blocks, with changes between said first consistent state and said second consistent state being stored in said buffers and in ones of said second set of blocks not pointed to by said on-disk inode.	
10. A device as in claim 9, wherein said file system on said storage system always moves atomically from said first consistent state to said second consistent state.	Rosenblum teaches a checkpoint, which marks a consistent state of the file system. Rosenblum, Sec. 4.4.1, p. 9. Once all modifications are written to disk, a checkpoint region is written, atomically moving the file system to a new consistent state. <i>Id.</i> The checkpoint region points to the last segment written and the location of the inode map. <i>Id.</i>
11. A device as in claim 10, wherein said file system on said storage system moves atomically to said second consistent state by flushing said changes from said buffers to said storage system before updating said on-disk root inode with information from said incore root inode.	Rosenblum teaches a checkpoint, which marks a consistent state of the file system. Rosenblum, Sec. 4.4.1, p. 9. Once all modifications are written to disk, a checkpoint region is written, atomically moving the file system to a new consistent state. <i>Id.</i> The checkpoint region points to the last segment written and the location of the inode map. <i>Id.</i>
12. A device as in claim 11, wherein updating said on-disk root inode further comprises updating said on-disk root inode and then a copy of said on-disk root inode such that if updating said on-disk root inode is interrupted, said copy of said on-disk root inode still points to said first consistent state of said file system.	Replicating the metadata, such as a root inode, is a technique that is well known in the art. For instance, Leffler teaches that in the Unix filesystem the superblock is replicated to allow recovery from crashes that corrupt the primary copy of the superblock. Leffler, Sec. 7.3, p. 196. In view of the knowledge in the art and, it would have been obvious for one of ordinary skill to make a copy of the root inode to recover from crashes that corrupt the primary copy of the root inode.
13. A device as in claim 9, wherein the instructions further comprise the step of creating a snapshot of said file system by copying only said on-disk root inode.	Rosenblum teaches that the group commit concept at the basis of atomic updating taught in Rosenblum is found database literature. Rosenblum, Sec. 4.5, p. 10. Therefore, one of ordinary skill in the art would find motivation to look to the teachings of database art, such as Ylonen. Ylonen teaches an atomic commit method for databases, wherein a page table pointer (equivalent to a root inode) is updated to atomically advance the database to the next consistent state. Ylonen, Sec. 2,

	pp. 1-2, Fig. 1. Ylonen further teaches taking one or more snapshots by saving the address of the page table and preventing freeing of data pages. Ylonen, Sec. 8, p. 4, Fig. 2. The page table address is in the page table pointer. Thus, every time the page table pointer is copied (an operation that occurs at every commit cycle), “[w]e can have an arbitrary number of snapshots at different times, and they will all be consistent as long as their pages are not freed.” <i>Id.</i>
14. A device as in claim 13, wherein when said snapshot is created, said snapshot and said file system share said first set of blocks on said storage system.	<p>Ylonen teaches a shadow paging (copy-on-write) technique that “never modifies accessible pages,” therefore a snapshot and the active file system share unmodified blocks. Ylonen, Sec. 8, p. 4, Figs. 1 and 2.</p>  <p>Figure 2: Snapshots represent consistent database states as of some time in the past.</p>
15. A device as in claim 9, wherein the instructions further comprise the step of creating plural snapshots of said file system by copying only said on-disk root inode at different times.	Ylonen teaches an atomic commit method for databases, wherein a page table pointer (equivalent to a root inode) is updated to atomically advance the database to the next consistent state. Ylonen, Sec. 2, pp. 1-2, Fig. 1. Ylonen further teaches taking one or more snapshots by saving the address of the page table and preventing freeing of data pages. Ylonen, Sec. 8, p. 4, Fig. 2. The page table address is in the page table pointer. Thus, every time the page table pointer is copied (an operation that occurs at every commit cycle), “[w]e can have an arbitrary number of snapshots at different times, and they will all be consistent as long as their pages are not freed.” <i>Id.</i>
16. A device as in claim 15, wherein when each one of said plural snapshots is created, each one of said snapshots and said file system share said first set of blocks on said storage system.	Ylonen teaches a shadow paging (copy-on-write) technique that “never modifies accessible pages,” therefore a snapshot and the active file system share unmodified blocks. Ylonen, Sec. 8, p. 4, Figs. 1 and 2.

	 <p>Figure 2: Snapshots represent consistent database states as of some time in the past.</p>
<p>17. An article of manufacture comprising a machine-readable storage medium storing instructions executable by a processor coupled to a memory and to a storage system, said storage system comprising one or more hard disks, said memory and said storage system storing a file system, wherein the instructions, when executed by the processor, cause the processor to (a) maintain an on-disk root inode on said storage system, said on-disk root inode pointing directly and indirectly to a first set of blocks on said storage system that store a first consistent state of said file system, and (b) maintain an incore root inode in said memory, said incore root inode pointing directly and indirectly to buffers in said memory and a second set of blocks on said storage system, said buffers and said second set of blocks storing data and metadata for a second consistent state of said file system, said second set of blocks including at least some blocks in said first set of blocks, with changes between said first consistent state and said second consistent state being stored in said buffers and in ones of said second set of blocks not pointed to by said on-disk inode.</p>	<p>The file system in Rosenblum is inherently stored on machine readable medium in the form of instructions executable by a processor coupled to a memory and one or more hard disks.</p> <p>Rosenblum maintains an on disk root inode. The format of the inodes and indirect blocks is the same as in the standard Unix file system, which is known to one of ordinary skill to provide for root inodes that directly and indirectly point to data blocks. <i>See Rosenblum, Sec. 4.2.1, p.6.</i></p> <p>Rosenblum further teaches that file system metadata, including inodes, and data blocks are cached in memory, where changes are accumulated before being forced to disk. Rosenblum, Sec. 4.1, p. 5. Before being forced to disk, the modified in-core root inode inherently points to buffers in memory containing modified data, new blocks already forced to disk, and old blocks with unmodified data.</p>
<p>18. An article of manufacture as in claim 17, wherein the instructions further cause the processor to move atomically said file system on said storage system from said first consistent state to said second</p>	<p>Rosenblum teaches a checkpoint, which marks a consistent state of the file system. Rosenblum, Sec. 4.4.1, p. 9. Once all modifications are written to disk, a checkpoint region is written, atomically moving the file system to a new consistent state. <i>Id.</i> The checkpoint region points to the last segment written and the</p>

consistent state.	location of the inode map. <i>Id.</i>
19. An article of manufacture as in claim 18, wherein the instructions cause the processor to move atomically said file system on said storage system to said second consistent state by flushing said changes from said buffers to said storage system before updating said on-disk root inode with information from said incore root inode.	Rosenblum teaches a checkpoint, which marks a consistent state of the file system. Rosenblum, Sec. 4.4.1, p. 9. Once all modifications are written to disk, a checkpoint region is written, atomically moving the file system to a new consistent state. <i>Id.</i> The checkpoint region points to the last segment written and the location of the inode map. <i>Id.</i>
20. An article of manufacture as in claim 19, wherein updating said on-disk root inode comprises updating said on-disk root inode and then a copy of said on-disk root inode such that if updating said on-disk root inode is interrupted, said copy of said on-disk root inode still points to said first consistent state of said file system.	Replicating the metadata, such as a root inode, is a technique that is well known in the art. For instance, Leffler teaches that in the Unix filesystem the superblock is replicated to allow recovery from crashes that corrupt the primary copy of the superblock. Leffler, Sec. 7.3, p. 196. In view of the knowledge in the art and, it would have been obvious for one of ordinary skill to make a copy of the root inode to recover from crashes that corrupt the primary copy of the root inode.
21. An article of manufacture as in claim 17, wherein the instructions further cause the processor to create a snapshot of said file system by copying only said on-disk root inode.	Rosenblum teaches that the group commit concept at the basis of atomic updating taught in Rosenblum is found database literature. Rosenblum, Sec. 4.5, p. 10. Therefore, one of ordinary skill in the art would find motivation to look to the teachings of database art, such as Ylonen. Ylonen teaches an atomic commit method for databases, wherein a page table pointer (equivalent to a root inode) is updated to atomically advance the database to the next consistent state. Ylonen, Sec. 2, pp. 1-2, Fig. 1. Ylonen further teaches taking one or more snapshots by saving the address of the page table and preventing freeing of data pages. Ylonen, Sec. 8, p. 4, Fig. 2. The page table address is in the page table pointer. Thus, every time the page table pointer is copied (an operation that occurs at every commit cycle), “[w]e can have an arbitrary number of snapshots at different times, and they will all be consistent as long as their pages are not freed.” <i>Id.</i>
22. An article of manufacture as in claim 21, wherein the instructions cause the processor to create said	Ylonen teaches a shadow paging (copy-on-write) technique that “never modifies accessible pages,” therefore a snapshot and the active file system share

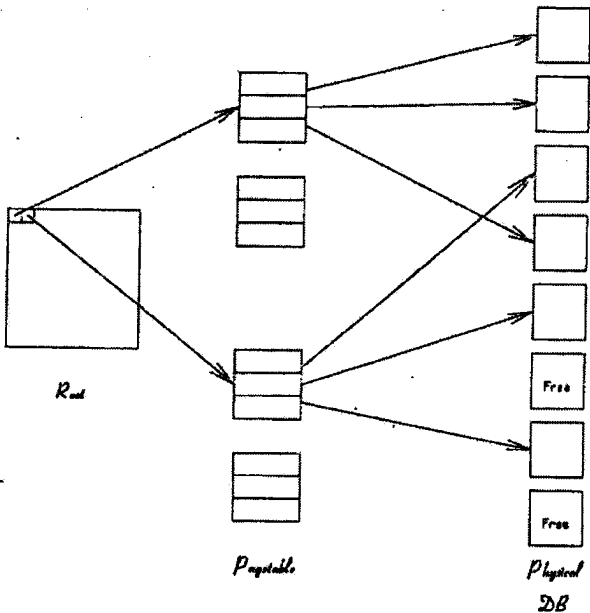
<p>snapshot so that said snapshot and said file system share said first set of blocks on said storage system when said snapshot is created.</p>	<p>unmodified blocks. Ylonen, Sec. 8, p. 4, Figs. 1 and 2.</p>  <p>Figure 2: Snapshots represent consistent database states as of some time in the past.</p>
<p>23. An article of manufacture as in claim 17, wherein the instructions further cause the processor to create plural snapshots of said file system by copying only said on-disk root inode at different times.</p>	<p>Ylonen teaches an atomic commit method for databases, wherein a page table pointer (equivalent to a root inode) is updated to atomically advance the database to the next consistent state. Ylonen, Sec. 2, pp. 1-2, Fig. 1. Ylonen further teaches taking one or more snapshots by saving the address of the page table and preventing freeing of data pages. Ylonen, Sec. 8, p. 4, Fig. 2. The page table address is in the page table pointer. Thus, every time the page table pointer is copied (an operation that occurs at every commit cycle), “[w]e can have an arbitrary number of snapshots at different times, and they will all be consistent as long as their pages are not freed.” <i>Id.</i></p>
<p>24. An article of manufacture as in claim 23, wherein the instructions, when executed by the processor, cause the processor to create each one of said plural snapshots so that each one of said snapshots and said file system share said first set of blocks on said storage system when each one of said plural snapshots is created.</p>	<p>Ylonen teaches a shadow paging (copy-on-write) technique that “never modifies accessible pages,” therefore a snapshot and the active file system share unmodified blocks. Ylonen, Sec. 8, p. 4, Figs. 1 and 2.</p>  <p>Figure 2: Snapshots represent consistent database states as of some time in the past.</p>

### Seventh Basis of Invalidity


The references applicable to the seventh basis of invalidity are:


1. Kent, *Performance and Implementation Issues in Database Crash Recovery*, Ph.D. Dissertation, Princeton University, 1985 (“Kent”).
2. Popek, as cited hereinabove.
3. Ylonen, as cited hereinabove.

The pertinence and manner of applying Kent, Popek and Ylonen to claims 1- 3, 5-11, 13-19, 21-24 for which re-examination is requested is as follows:

Claims of '211 Patent	Kent and Popek and Ylonen
<p>1. A method of maintaining a file system stored in a memory and on a storage system that includes one or more hard disks, said method comprising steps of:</p>	<p>Kent discloses a database using page shadowing (copy-on-write) and atomic commit to advance the database from one consistent state to another. Kent, Sec. 3.5, pp. 28-31, 40-43. The Kent database inherently resides on one or more hard disks. As is understood by one of ordinary skill in the art, databases and file systems are closely related fields of art. For example, Popek teaches that "[t]he important concept of atomically committing changes has been imported from the database world and integrated into LOCUS." Popek, Sec. 3.4.6, p. 46. Therefore, one of ordinary skill in the art would find motivation to look to the teachings of database art, such as Kent and Ylonen to implement a file system.</p>
<p>maintaining an on-disk root inode on said storage system, said on-disk root inode pointing directly and indirectly to a first set of blocks on said storage system that store a first consistent state of said file system; and</p>	<p>Kent database structures include the root, which points to the global version of each mapping page: "That is, the root defines the most recent, consistent state of the page table (Just as the page table defined the most recent, consistent state of the logical database.)" Kent, p. 41, Fig. 3-4.</p>  <p>The diagram illustrates the Kent database structure. On the left, a box labeled 'Root' has two arrows pointing to two separate 'Page table' structures in the center. Each 'Page table' is represented by a vertical stack of three boxes. Arrows from these 'Page table' structures point to a 'Physical DB' on the right, which is a vertical stack of six boxes. The top four boxes of the 'Physical DB' are connected to the 'Page table' structures, while the bottom two are labeled 'Free'.</p> <p style="text-align: center;">Figure 3-4</p> <p>The structures shown in Fig. 3-4 would be understood</p>

	by one of ordinary skill in the art to be equivalent to root inode, intermediate inodes, and data blocks in a file system. These structures reside on disk and in-core in cache memory. <i>See e.g. Kent, pp. 28-31, 89-90.</i>
maintaining an incore root inode in said memory, said incore root inode pointing directly and indirectly to buffers in said memory and a second set of blocks on said storage system, said buffers and said second set of blocks storing data and metadata for a second consistent state of said file system, said second set of blocks including at least some blocks in said first set of blocks, with changes between said first consistent state and said second consistent state being stored in said buffers and in ones of said second set of blocks not pointed to by said on-disk inode.	The cache memory in Kent contains buffers that hold modified data blocks. Kent, p. 90. Kent describes a number of “primitives” – operations that allow the shadowing algorithm CRM <sub>shadow</sub> to “maintain[] buffers and . . . flush[] them to disk.” Kent, Sec. 3.5.2, p. 34. The structure shown in Fig. 3-4 is held in-core, with the root pointing directly and indirectly to buffers storing metadata (such as page tables) and data. Before it is flushed to disk, the root also points to updated blocks on disk. Kent, Sec. 3.5.2, pp. 40-43. Kent teaches an atomic commit operation by flushing the root from cache memory to disk: “The final step (flushing the root) must be executed atomically, as it defines the line between transaction commit and abort. To commit, a transaction first flushes the subtree that corresponds to its modifications, and then the root.” Kent, Sec. 3.5.2, pp. 42-43, pp. 46-47.
2. A method as in claim 1, wherein said file system on said storage system always moves atomically from said first consistent state to said second consistent state.	Kent teaches an atomic commit operation by flushing the root from cache memory to disk: “The final step (flushing the root) must be executed atomically, as it defines the line between transaction commit and abort. To commit, a transaction first flushes the subtree that corresponds to its modifications, and then the root.” Kent, Sec. 3.5.2, pp. 42-43, pp. 46-47.
3. A method as in claim 2, wherein said file system on said storage system moves atomically to said second consistent state by flushing said changes from said buffers to said storage system before updating said on-disk root inode with information from said incore root inode.	Kent teaches an atomic commit operation by flushing the root from cache memory to disk: “The final step (flushing the root) must be executed atomically, as it defines the line between transaction commit and abort. To commit, a transaction first flushes the subtree that corresponds to its modifications, and then the root.” Kent, Sec. 3.5.2, pp. 42-43, pp. 46-47.
5. A method as in claim 1, further comprising the step of creating a snapshot of said file system by copying only said on-disk root inode.	As Kent Fig. 3-4 shows, the root points to the current database (top pointer) as well as previous consistent states of the database, i.e. snapshots (bottom pointer). So long as the older versions of data and metadata pages are not released, snapshots are accessible through the root and are created by copying and updating the root. Creating snapshots through updating a page table

	<p>pointer is further disclosed and described in Ylonen, which refers to Kent, thereby motivating one of ordinary skill to combine the teachings of Ylonen with the teachings of Kent. Ylonen, p. 8. Ylonen teaches an atomic commit method for databases, wherein a page table pointer (equivalent to a root inode) is updated to atomically advance the database to the next consistent state. Ylonen, Sec. 2, pp. 1-2, Fig. 1. Ylonen further teaches taking one or more snapshots by saving the address of the page table and preventing freeing of data pages. Ylonen, Sec. 8, p. 4, Fig. 2. The page table address is in the page table pointer. Thus, every time the page table pointer is copied (an operation that occurs at every commit cycle), “[w]e can have an arbitrary number of snapshots at different times, and they will all be consistent as long as their pages are not freed.” <i>Id.</i> Ylonen further teaches using a master pointer technique, adding another level of indirection above the page table pointers of Kent and Ylonen: “One possible implementation is to have an atomically updatable master pointer instead of a page table pointer. The master pointer would point to several page table pointer pages (remember, the page table pointer may also contain other data in addition to the actual address of the page table). The page table pointers would reside in normal shadowed database storage.” Ylonen, Sec. 10, p. 4.</p>
6. A method as in claim 5, wherein when said snapshot is created, said snapshot and said file system share said first set of blocks on said storage system.	<p>Ylonen teaches a shadow paging (copy-on-write) technique that “never modifies accessible pages,” therefore a snapshot and the active file system share unmodified blocks. Ylonen, Sec. 8, p. 4, Figs. 1 and 2.</p>  <p>Figure 2: Snapshots represent consistent database states as of some time in the past.</p>
7. A method as in claim 1, further comprising the step of creating plural snapshots of said file system by copying only said on-disk root inode at different times.	<p>Ylonen teaches an atomic commit method for databases, wherein a page table pointer (equivalent to a root inode) is updated to atomically advance the database to the next consistent state. Ylonen, Sec. 2, pp. 1-2, Fig. 1. Ylonen further teaches taking one or more snapshots by saving the address of the page table and preventing</p>

	freeing of data pages. Ylonen, Sec. 8, p. 4, Fig. 2. The page table address is in the page table pointer. Thus, every time the page table pointer is copied (an operation that occurs at every commit cycle), “[w]e can have an arbitrary number of snapshots at different times, and they will all be consistent as long as their pages are not freed.” <i>Id.</i> Ylonen further teaches using a master pointer technique, adding another level of indirection above the page table pointers of Kent and Ylonen: “One possible implementation is to have an atomically updatable master pointer instead of a page table pointer. The master pointer would point to several page table pointer pages (remember, the page table pointer may also contain other data in addition to the actual address of the page table). The page table pointers would reside in normal shadowed database storage.” Ylonen, Sec. 10, p. 4.
8. A method as in claim 7, wherein when each one of said plural snapshots is created, each one of said snapshots and said file system share said first set of blocks on said storage system.	<p>Ylonen teaches a shadow paging (copy-on-write) technique that “never modifies accessible pages,” therefore a snapshot and the active file system share unmodified blocks. Ylonen, Sec. 8, p. 4, Figs. 1 and 2.</p>  <p>Figure 2: Snapshots represent consistent database states as of some time in the past.</p>
9. A device comprising: a processor; a memory; and	Kent teaches a database that is inherently executed on a processor and a memory.
a storage system including one or more hard disks;	The Kent database is inherently stored on one or more hard disks.
wherein said memory and said storage system store a file system; and	The Kent database is stored on disk and in memory. <i>See e.g.</i> Kent, pp. 28-31, 89-90.
wherein said memory also stores information including instructions executable by said processor to maintain said file system, the instructions including steps of (a) maintaining an on-disk root inode on said storage system, said on-disk root inode pointing directly and indirectly to a first set of blocks on said storage system that store a first consistent	<p>The Kent database inherently exists as instructions executable by a processor.</p> <p>Kent discloses a database using page shadowing (copy-on-write) and atomic commit to advance the database from one consistent state to another. Kent, Sec. 3.5, pp. 28-31, 40-43. The Kent database inherently resides on one or more hard disks. As is understood by one of ordinary skill in the art, databases and file systems are closely related fields of art. For example, Popek teaches</p>

state of said file system, and (b) maintaining an incore root inode in said memory, said incore root inode pointing directly and indirectly to buffers in said memory and a second set of blocks on said storage system, said buffers and said second set of blocks storing data and metadata for a second consistent state of said file system, said second set of blocks including at least some blocks in said first set of blocks, with changes between said first consistent state and said second consistent state being stored in said buffers and in ones of said second set of blocks not pointed to by said on-disk inode.

that “[t]he important concept of atomically committing changes has been imported from the database world and integrated into LOCUS.” Popek, Sec. 3.4.6, p. 46. Therefore, one of ordinary skill in the art would find motivation to look to the teachings of database art, such as Kent and Ylonen to implement a file system.

Kent database structures include the root, which points to the global version of each mapping page: “That is, the root defines the most recent, consistent state of the page table (Just as the page table defined the most recent, consistent state of the logical database.)” Kent, p. 41, Fig. 3-4.

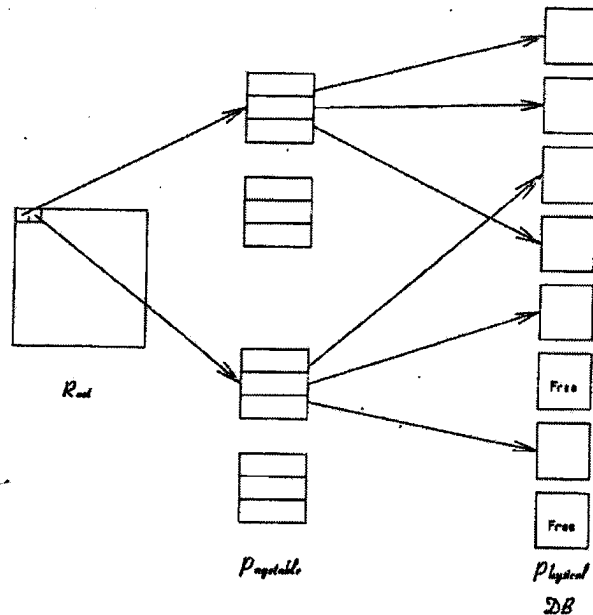




Figure 3-4

The structures shown in Fig. 3-4 would be understood by one of ordinary skill in the art to be equivalent to root inode, intermediate inodes, and data blocks in a file system. These structures reside on disk and in-core in cache memory. See e.g. Kent, pp. 28-31, 89-90.

The cache memory in Kent contains buffers that hold modified data blocks. Kent, p. 90. Kent describes a number of “primitives” – operations that allow the shadowing algorithm CRM<sub>shadow</sub> to “maintain[] buffers and . . . flush[] them to disk.” Kent, Sec. 3.5.2, p. 34.

	<p>The structure shown in Fig. 3-4 is held in-core, with the root pointing directly and indirectly to buffers storing metadata (such as page tables) and data. Before it is flushed to disk, the root also points to updated blocks on disk. Kent, Sec. 3.5.2, pp. 40-43. Kent teaches an atomic commit operation by flushing the root from cache memory to disk: "The final step (flushing the root) must be executed atomically, as it defines the line between transaction commit and abort. To commit, a transaction first flushes the subtree that corresponds to its modifications, and then the root." Kent, Sec. 3.5.2, pp. 42-43, pp. 46-47.</p>
10. A device as in claim 9, wherein said file system on said storage system always moves atomically from said first consistent state to said second consistent state.	<p>Kent teaches an atomic commit operation by flushing the root from cache memory to disk: "The final step (flushing the root) must be executed atomically, as it defines the line between transaction commit and abort. To commit, a transaction first flushes the subtree that corresponds to its modifications, and then the root." Kent, Sec. 3.5.2, pp. 42-43, pp. 46-47.</p>
11. A device as in claim 10, wherein said file system on said storage system moves atomically to said second consistent state by flushing said changes from said buffers to said storage system before updating said on-disk root inode with information from said incore root inode.	<p>Kent teaches an atomic commit operation by flushing the root from cache memory to disk: "The final step (flushing the root) must be executed atomically, as it defines the line between transaction commit and abort. To commit, a transaction first flushes the subtree that corresponds to its modifications, and then the root." Kent, Sec. 3.5.2, pp. 42-43, pp. 46-47.</p>
13. A device as in claim 9, wherein the instructions further comprise the step of creating a snapshot of said file system by copying only said on-disk root inode.	<p>As Kent Fig. 3-4 shows, the root points to the current database (top pointer) as well as previous consistent states of the database, i.e. snapshots (bottom pointer). So long as the older versions of data and metadata pages are not released, snapshots are accessible through the root and are created by copying and updating the root. Creating snapshots through updating a page table pointer is further disclosed and described in Ylonen, which refers to Kent, thereby motivating one of ordinary skill to combine the teachings of Ylonen with the teachings of Kent. Ylonen, p. 8. Ylonen teaches an atomic commit method for databases, wherein a page table pointer (equivalent to a root inode) is updated to atomically advance the database to the next consistent state. Ylonen, Sec. 2, pp. 1-2, Fig. 1. Ylonen further teaches taking one or more snapshots by saving the</p>

	<p>address of the page table and preventing freeing of data pages. Ylonen, Sec. 8, p. 4, Fig. 2. The page table address is in the page table pointer. Thus, every time the page table pointer is copied (an operation that occurs at every commit cycle), “[w]e can have an arbitrary number of snapshots at different times, and they will all be consistent as long as their pages are not freed.” <i>Id.</i> Ylonen further teaches using a master pointer technique, adding another level of indirection above the page table pointers of Kent and Ylonen: “One possible implementation is to have an atomically updatable master pointer instead of a page table pointer. The master pointer would point to several page table pointer pages (remember, the page table pointer may also contain other data in addition to the actual address of the page table). The page table pointers would reside in normal shadowed database storage.” Ylonen, Sec. 10, p. 4.</p>
<p>14. A device as in claim 13, wherein when said snapshot is created, said snapshot and said file system share said first set of blocks on said storage system.</p>	<p>Ylonen teaches a shadow paging (copy-on-write) technique that “never modifies accessible pages,” therefore a snapshot and the active file system share unmodified blocks. Ylonen, Sec. 8, p. 4, Figs. 1 and 2.</p>  <p>Figure 2: Snapshots represent consistent database states as of some time in the past.</p>
<p>15. A device as in claim 9, wherein the instructions further comprise the step of creating plural snapshots of said file system by copying only said on-disk root inode at different times.</p>	<p>Ylonen teaches an atomic commit method for databases, wherein a page table pointer (equivalent to a root inode) is updated to atomically advance the database to the next consistent state. Ylonen, Sec. 2, pp. 1-2, Fig. 1. Ylonen further teaches taking one or more snapshots by saving the address of the page table and preventing freeing of data pages. Ylonen, Sec. 8, p. 4, Fig. 2. The page table address is in the page table pointer. Thus, every time the page table pointer is copied (an operation that occurs at every commit cycle), “[w]e can have an arbitrary number of snapshots at different times, and they will all be consistent as long as their pages are not freed.” <i>Id.</i> Ylonen further teaches using a master pointer technique, adding another level of indirection above the page table pointers of Kent and Ylonen: “One</p>

	possible implementation is to have an atomically updatable master pointer instead of a page table pointer. The master pointer would point to several page table pointer pages (remember, the page table pointer may also contain other data in addition to the actual address of the page table). The page table pointers would reside in normal shadowed database storage.” Ylonen, Sec. 10, p. 4.
16. A device as in claim 15, wherein when each one of said plural snapshots is created, each one of said snapshots and said file system share said first set of blocks on said storage system.	<p>Ylonen teaches a shadow paging (copy-on-write) technique that “never modifies accessible pages,” therefore a snapshot and the active file system share unmodified blocks. Ylonen, Sec. 8, p. 4, Figs. 1 and 2.</p>  <p>Figure 2: Snapshots represent consistent database states as of some time in the past.</p>
17. An article of manufacture comprising a machine-readable storage medium storing instructions executable by a processor coupled to a memory and to a storage system, said storage system comprising one or more hard disks, said memory and said storage system storing a file system, wherein the instructions, when executed by the processor, cause the processor to (a) maintain an on-disk root inode on said storage system, said on-disk root inode pointing directly and indirectly to a first set of blocks on said storage system that store a first consistent state of said file system, and (b) maintain an incore root inode in said memory, said incore root inode pointing directly and indirectly to buffers in said memory and a second set of blocks on said storage system, said buffers and said second set of blocks storing data and metadata for a second consistent state of said file	<p>The Kent database inherently exists as instructions executable by a processor coupled with a memory and a storage system comprising one or more hard disks.</p> <p>Kent discloses a database using page shadowing (copy-on-write) and atomic commit to advance the database from one consistent state to another. Kent, Sec. 3.5, pp. 28-31, 40-43. The Kent database inherently resides on one or more hard disks. As is understood by one of ordinary skill in the art, databases and file systems are closely related fields of art. For example, Popek teaches that “[t]he important concept of atomically committing changes has been imported from the database world and integrated into LOCUS.” Popek, Sec. 3.4.6, p. 46. Therefore, one of ordinary skill in the art would find motivation to look to the teachings of database art, such as Kent and Ylonen to implement a file system.</p> <p>Kent database structures include the root, which points to the global version of each mapping page: “That is, the root defines the most recent, consistent state of the page table (Just as the page table defined the most recent, consistent state of the logical database.)” Kent, p. 41, Fig. 3-4.</p>

system, said second set of blocks including at least some blocks in said first set of blocks, with changes between said first consistent state and said second consistent state being stored in said buffers and in ones of said second set of blocks not pointed to by said on-disk inode.

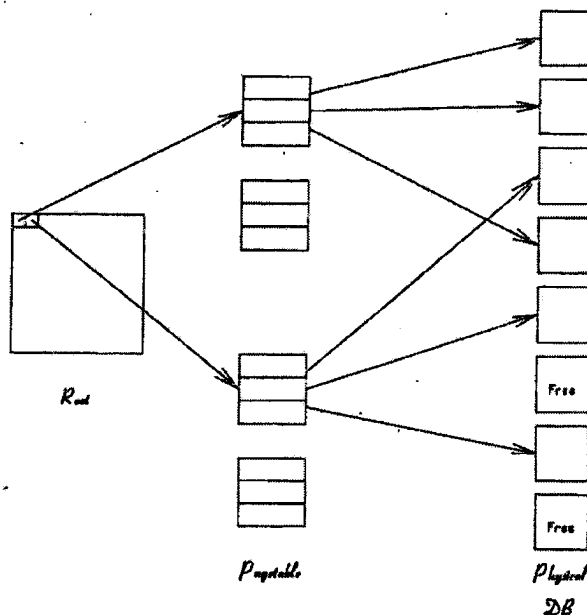


Figure 3-4


The structures shown in Fig. 3-4 would be understood by one of ordinary skill in the art to be equivalent to root inode, intermediate inodes, and data blocks in a file system. These structures reside on disk and in-core in cache memory. *See e.g.* Kent, pp. 28-31, 89-90.

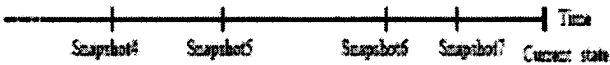
The cache memory in Kent contains buffers that hold modified data blocks. Kent, p. 90. Kent describes a number of “primitives” – operations that allow the shadowing algorithm  $CRM_{shadow}$  to “maintain[] buffers and . . . flush[] them to disk.” Kent, Sec. 3.5.2, p. 34. The structure shown in Fig. 3-4 is held in-core, with the root pointing directly and indirectly to buffers storing metadata (such as page tables) and data. Before it is flushed to disk, the root also points to updated blocks on disk. Kent, Sec. 3.5.2, pp. 40-43. Kent teaches an atomic commit operation by flushing the root from cache memory to disk: “The final step (flushing the root) must be executed atomically, as it defines the line between transaction commit and abort. To commit, a transaction first flushes the subtree that corresponds to its modifications, and then the root.” Kent, Sec. 3.5.2, pp. 42-43, pp. 46-47.

18. An article of manufacture as in

Kent teaches an atomic commit operation by flushing

claim 17, wherein the instructions further cause the processor to move atomically said file system on said storage system from said first consistent state to said second consistent state.	the root from cache memory to disk: "The final step (flushing the root) must be executed atomically, as it defines the line between transaction commit and abort. To commit, a transaction first flushes the subtree that corresponds to its modifications, and then the root." Kent, Sec. 3.5.2, pp. 42-43, pp. 46-47.
19. An article of manufacture as in claim 18, wherein the instructions cause the processor to move atomically said file system on said storage system to said second consistent state by flushing said changes from said buffers to said storage system before updating said on-disk root inode with information from said incore root inode.	Kent teaches an atomic commit operation by flushing the root from cache memory to disk: "The final step (flushing the root) must be executed atomically, as it defines the line between transaction commit and abort. To commit, a transaction first flushes the subtree that corresponds to its modifications, and then the root." Kent, Sec. 3.5.2, pp. 42-43, pp. 46-47.
21. An article of manufacture as in claim 17, wherein the instructions further cause the processor to create a snapshot of said file system by copying only said on-disk root inode.	As Kent Fig. 3-4 shows, the root points to the current database (top pointer) as well as previous consistent states of the database, i.e. snapshots (bottom pointer). So long as the older versions of data and metadata pages are not released, snapshots are accessible through the root and are created by copying and updating the root. Creating snapshots through updating a page table pointer is further disclosed and described in Ylonen, which refers to Kent, thereby motivating one of ordinary skill to combine the teachings of Ylonen with the teachings of Kent. Ylonen, p. 8. Ylonen teaches an atomic commit method for databases, wherein a page table pointer (equivalent to a root inode) is updated to atomically advance the database to the next consistent state. Ylonen, Sec. 2, pp. 1-2, Fig. 1. Ylonen further teaches taking one or more snapshots by saving the address of the page table and preventing freeing of data pages. Ylonen, Sec. 8, p. 4, Fig. 2. The page table address is in the page table pointer. Thus, every time the page table pointer is copied (an operation that occurs at every commit cycle), "[w]e can have an arbitrary number of snapshots at different times, and they will all be consistent as long as their pages are not freed." <i>Id.</i> Ylonen further teaches using a master pointer technique, adding another level of indirection above the page table pointers of Kent and Ylonen: "One possible implementation is to have an atomically updatable master pointer instead of a page table pointer. The

	<p>master pointer would point to several page table pointer pages (remember, the page table pointer may also contain other data in addition to the actual address of the page table). The page table pointers would reside in normal shadowed database storage.” Ylonen, Sec. 10, p. 4.</p>
<p>22. An article of manufacture as in claim 21, wherein the instructions cause the processor to create said snapshot so that said snapshot and said file system share said first set of blocks on said storage system when said snapshot is created.</p>	<p>Ylonen teaches a shadow paging (copy-on-write) technique that “never modifies accessible pages,” therefore a snapshot and the active file system share unmodified blocks. Ylonen, Sec. 8, p. 4, Figs. 1 and 2.</p>  <p>Figure 2: Snapshots represent consistent database states as of some time in the past.</p>
<p>23. An article of manufacture as in claim 17, wherein the instructions further cause the processor to create plural snapshots of said file system by copying only said on-disk root inode at different times.</p>	<p>Ylonen teaches an atomic commit method for databases, wherein a page table pointer (equivalent to a root inode) is updated to atomically advance the database to the next consistent state. Ylonen, Sec. 2, pp. 1-2, Fig. 1. Ylonen further teaches taking one or more snapshots by saving the address of the page table and preventing freeing of data pages. Ylonen, Sec. 8, p. 4, Fig. 2. The page table address is in the page table pointer. Thus, every time the page table pointer is copied (an operation that occurs at every commit cycle), “[w]e can have an arbitrary number of snapshots at different times, and they will all be consistent as long as their pages are not freed.” <i>Id.</i> Ylonen further teaches using a master pointer technique, adding another level of indirection above the page table pointers of Kent and Ylonen: “One possible implementation is to have an atomically updatable master pointer instead of a page table pointer. The master pointer would point to several page table pointer pages (remember, the page table pointer may also contain other data in addition to the actual address of the page table). The page table pointers would reside in normal shadowed database storage.” Ylonen, Sec. 10, p. 4.</p>
<p>24. An article of manufacture as in claim 23, wherein the instructions, when executed by the processor,</p>	<p>Ylonen teaches a shadow paging (copy-on-write) technique that “never modifies accessible pages,” therefore a snapshot and the active file system share</p>

cause the processor to create each one of said plural snapshots so that each one of said snapshots and said file system share said first set of blocks on said storage system when each one of said plural snapshots is created.	<p>unmodified blocks. Ylonen, Sec. 8, p. 4, Figs. 1 and 2.</p>  <p>Figure 2: Snapshots represent consistent database states as of some time in the past.</p>
--	--

### Eighth Basis of Invalidity

The references applicable to the eighth basis of invalidity are:

1. Rosenblum, Ousterhout, *The Design and Implementation of a Log-Structured File System*, Proceedings of the 13th ACM Symposium on OS Principles, 1991 (“Rosenblum and Ousterhout”)
2. Leffler, as cited hereinabove.

The pertinence and manner of applying Rosenblum and Ousterhout and Leffler to claims 1- 6, 9-14 and 17-22 for which re-examination is requested is as follows:

<b>Claims of ‘211 Patent</b>	<b>Rosenblum and Ousterhout and Leffler</b>
1. A method of maintaining a file system stored in a memory and on a storage system that includes one or more hard disks, said method comprising steps of:	Rosenblum and Ousterhout teaches a file system stored in memory and on hard disks.
maintaining an on-disk root inode on said storage system, said on-disk root inode pointing directly and indirectly to a first set of blocks on said storage system that store a first consistent state of said file system; and	Rosenblum and Ousterhout maintains an on disk root inode. The format of the inodes and indirect blocks is the same as in the standard Unix file system, which is known to one of ordinary skill to provide for root inodes that directly and indirectly point to data blocks. <i>See</i> Rosenblum and Ousterhout, Sec. 3.1, p. 6, <i>see also e.g.</i> Leffler, Fig. 7.6, p. 194.
maintaining an incore root inode in said memory, said incore root inode pointing directly and indirectly to buffers in said memory and a second set of blocks on said storage system, said buffers and said second set of blocks storing data and metadata for a	Rosenblum and Ousterhout teaches that file system metadata, including inodes, and data blocks are cached in memory, where changes are accumulated before being forced to disk. Rosenblum and Ousterhout, Sec. 3, p. 3. Before being forced to disk, the modified incore root inode inherently points to buffers in memory containing modified data, modified blocks, and old

second consistent state of said file system, said second set of blocks including at least some blocks in said first set of blocks, with changes between said first consistent state and said second consistent state being stored in said buffers and in ones of said second set of blocks not pointed to by said on-disk inode.	blocks with unmodified data.
2. A method as in claim 1, wherein said file system on said storage system always moves atomically from said first consistent state to said second consistent state.	Rosenblum and Ousterhout teaches a checkpoint, which marks a consistent state of the file system. Rosenblum and Ousterhout, Sec. 3, p. 3, Sec. 4.1, p. 9. Once all modifications are written to disk, a checkpoint region is written, atomically moving the file system to a new consistent state. <i>Id.</i> The checkpoint region points to the last segment written and the location of the inode map. <i>Id.</i>
3. A method as in claim 2, wherein said file system on said storage system moves atomically to said second consistent state by flushing said changes from said buffers to said storage system before updating said on-disk root inode with information from said incore root inode.	Rosenblum and Ousterhout teaches a checkpoint, which marks a consistent state of the file system. Rosenblum and Ousterhout, Sec. 3, p. 3, Sec. 4.1, p. 9. Once all modifications are written to disk, a checkpoint region is written, atomically moving the file system to a new consistent state. <i>Id.</i> The checkpoint region points to the last segment written and the location of the inode map. <i>Id.</i>
4. A method as in claim 3, wherein updating said on-disk root inode further comprises updating said on-disk root inode and then a copy of said on-disk root inode such that if updating said on-disk root inode is interrupted, said copy of said on-disk root inode still points to said first consistent state of said file system.	Replicating the metadata, such as a root inode, is a technique that is well known in the art. For instance, Leffler teaches that in the Unix filesystem the superblock is replicated to allow recovery from crashes that corrupt the primary copy of the superblock. Leffler, Sec. 7.3, p. 196. Furthermore, Rosenblum and Ousterhout teaches that the checkpoint region is replicated: "In order to handle a crash during a checkpoint operation there are actually two checkpoint regions, and checkpoint operation alternates between them." Rosenblum and Ousterhout, Sec. 4.1, p. 9. In view of the knowledge in the art and the teachings of Leffler and Rosenblum and Ousterhout, it would have been obvious for one of ordinary skill to make a copy of the root inode to recover from crashes that corrupt the primary copy of the root inode.
5. A method as in claim 1, further	Rosenblum and Ousterhout teaches a checkpoint, which

comprising the step of creating a snapshot of said file system by copying only said on-disk root inode.	marks a consistent state of the file system. Rosenblum and Ousterhout, Sec. 3, p. 3, Sec. 4.1, p. 9. Once all modifications are written to disk, a checkpoint region is written, atomically moving the file system to a new consistent state. <i>Id.</i> The checkpoint region points to the last segment written and the location of the inode map. <i>Id.</i> The checkpoint represents a snapshot of the file system state at the time of the checkpoint.
6. A method as in claim 5, wherein when said snapshot is created, said snapshot and said file system share said first set of blocks on said storage system.	Rosenblum and Ousterhout teaches a checkpoint, which marks a consistent state of the file system. Rosenblum and Ousterhout, Sec. 3, p. 3, Sec. 4.1, p. 9. Once all modifications are written to disk, a checkpoint region is written, atomically moving the file system to a new consistent state. <i>Id.</i> The checkpoint region points to the last segment written and the location of the inode map. <i>Id.</i> The checkpoint represents a snapshot of the file system state at the time of the checkpoint. The LFS file system does not overwrite old data blocks, instead data is written out in a log format. <i>See e.g.</i> Rosenblum and Ousterhout, Fig. 1 (left side showing LFS). Unmodified data blocks therefore are shared between the checkpoint and the live file system. <i>Id.</i>
9. A device comprising: a processor; a memory; and	The file system taught in Rosenblum and Ousterhout is inherently executed on a processor with a memory.
a storage system including one or more hard disks;	Rosenblum and Ousterhout teaches a file system using a cache memory and one or more disks.
wherein said memory and said storage system store a file system; and	The file system in Rosenblum and Ousterhout is stored in memory and on a storage system.
wherein said memory also stores information including instructions executable by said processor to maintain said file system, the instructions including steps of (a) maintaining an on-disk root inode on said storage system, said on-disk root inode pointing directly and indirectly to a first set of blocks on said storage system that store a first consistent state of said file system, and (b) maintaining an incore root inode in said memory, said incore root inode pointing directly and indirectly to buffers in said memory and a second set of blocks on said storage system,	<p>Rosenblum and Ousterhout maintains an on disk root inode. The format of the inodes and indirect blocks is the same as in the standard Unix file system, which is known to one of ordinary skill to provide for root inodes that directly and indirectly point to data blocks. <i>See</i> Rosenblum and Ousterhout, Sec. 3.1, p. 6, <i>see also e.g.</i> Leffler, Fig. 7.6, p. 194.</p> <p>Rosenblum and Ousterhout teaches that file system metadata, including inodes, and data blocks are cached in memory, where changes are accumulated before being forced to disk. Rosenblum and Ousterhout, Sec. 3, p. 3. Before being forced to disk, the modified incore root inode inherently points to buffers in memory containing modified data, modified blocks, and old blocks with unmodified data.</p>

<p>said buffers and said second set of blocks storing data and metadata for a second consistent state of said file system, said second set of blocks including at least some blocks in said first set of blocks, with changes between said first consistent state and said second consistent state being stored in said buffers and in ones of said second set of blocks not pointed to by said on-disk inode.</p>	
<p>10. A device as in claim 9, wherein said file system on said storage system always moves atomically from said first consistent state to said second consistent state.</p>	<p>Rosenblum and Ousterhout teaches a checkpoint, which marks a consistent state of the file system. Rosenblum and Ousterhout, Sec. 3, p. 3, Sec. 4.1, p. 9. Once all modifications are written to disk, a checkpoint region is written, atomically moving the file system to a new consistent state. <i>Id.</i> The checkpoint region points to the last segment written and the location of the inode map. <i>Id.</i></p>
<p>11. A device as in claim 10, wherein said file system on said storage system moves atomically to said second consistent state by flushing said changes from said buffers to said storage system before updating said on-disk root inode with information from said incore root inode.</p>	<p>Rosenblum and Ousterhout teaches a checkpoint, which marks a consistent state of the file system. Rosenblum and Ousterhout, Sec. 3, p. 3, Sec. 4.1, p. 9. Once all modifications are written to disk, a checkpoint region is written, atomically moving the file system to a new consistent state. <i>Id.</i> The checkpoint region points to the last segment written and the location of the inode map. <i>Id.</i></p>
<p>12. A device as in claim 11, wherein updating said on-disk root inode further comprises updating said on-disk root inode and then a copy of said on-disk root inode such that if updating said on-disk root inode is interrupted, said copy of said on-disk root inode still points to said first consistent state of said file system.</p>	<p>Replicating the metadata, such as a root inode, is a technique that is well known in the art. For instance, Leffler teaches that in the Unix filesystem the superblock is replicated to allow recovery from crashes that corrupt the primary copy of the superblock. Leffler, Sec. 7.3, p. 196. Furthermore, Rosenblum and Ousterhout teaches that the checkpoint region is replicated: "In order to handle a crash during a checkpoint operation there are actually two checkpoint regions, and checkpoint operation alternates between them." Rosenblum and Ousterhout, Sec. 4.1, p. 9. In view of the knowledge in the art and the teachings of Leffler and Rosenblum and Ousterhout, it would have been obvious for one of ordinary skill to make a copy of the root inode to recover from crashes that corrupt the primary copy of the root inode.</p>

<p>13. A device as in claim 9, wherein the instructions further comprise the step of creating a snapshot of said file system by copying only said on-disk root inode.</p>	<p>Rosenblum and Ousterhout teaches a checkpoint, which marks a consistent state of the file system. Rosenblum and Ousterhout, Sec. 3, p. 3, Sec. 4.1, p. 9. Once all modifications are written to disk, a checkpoint region is written, atomically moving the file system to a new consistent state. <i>Id.</i> The checkpoint region points to the last segment written and the location of the inode map. <i>Id.</i> The checkpoint represents a snapshot of the file system state at the time of the checkpoint.</p>
<p>14. A device as in claim 13, wherein when said snapshot is created, said snapshot and said file system share said first set of blocks on said storage system.</p>	<p>Rosenblum and Ousterhout teaches a checkpoint, which marks a consistent state of the file system. Rosenblum and Ousterhout, Sec. 3, p. 3, Sec. 4.1, p. 9. Once all modifications are written to disk, a checkpoint region is written, atomically moving the file system to a new consistent state. <i>Id.</i> The checkpoint region points to the last segment written and the location of the inode map. <i>Id.</i> The checkpoint represents a snapshot of the file system state at the time of the checkpoint. The LFS file system does not overwrite old data blocks, instead data is written out in a log format. <i>See e.g.</i> Rosenblum and Ousterhout, Fig. 1 (left side showing LFS). Unmodified data blocks therefore are shared between the checkpoint and the live file system. <i>Id.</i></p>
<p>17. An article of manufacture comprising a machine-readable storage medium storing instructions executable by a processor coupled to a memory and to a storage system, said storage system comprising one or more hard disks, said memory and said storage system storing a file system, wherein the instructions, when executed by the processor, cause the processor to (a) maintain an on-disk root inode on said storage system, said on-disk root inode pointing directly and indirectly to a first set of blocks on said storage system that store a first consistent state of said file system, and (b) maintain an incore root inode in said memory, said incore root inode pointing directly and indirectly to buffers in said memory and a second</p>	<p>The file system in Rosenblum and Ousterhout is inherently stored on machine readable medium in the form of instructions executable by a processor coupled to a memory and one or more hard disks.</p> <p>Rosenblum and Ousterhout maintains an on disk root inode. The format of the inodes and indirect blocks is the same as in the standard Unix file system, which is known to one of ordinary skill to provide for root inodes that directly and indirectly point to data blocks. <i>See</i> Rosenblum and Ousterhout, Sec. 3.1, p. 6, <i>see also e.g.</i> Leffler, Fig. 7.6, p. 194.</p> <p>Rosenblum and Ousterhout teaches that file system metadata, including inodes, and data blocks are cached in memory, where changes are accumulated before being forced to disk. Rosenblum and Ousterhout, Sec. 3, p. 3. Before being forced to disk, the modified incore root inode inherently points to buffers in memory containing modified data, modified blocks, and old</p>

set of blocks on said storage system, said buffers and said second set of blocks storing data and metadata for a second consistent state of said file system, said second set of blocks including at least some blocks in said first set of blocks, with changes between said first consistent state and said second consistent state being stored in said buffers and in ones of said second set of blocks not pointed to by said on-disk inode.	blocks with unmodified data.
18. An article of manufacture as in claim 17, wherein the instructions further cause the processor to move atomically said file system on said storage system from said first consistent state to said second consistent state.	Rosenblum and Ousterhout teaches a checkpoint, which marks a consistent state of the file system. Rosenblum and Ousterhout, Sec. 3, p. 3, Sec. 4.1, p. 9. Once all modifications are written to disk, a checkpoint region is written, atomically moving the file system to a new consistent state. <i>Id.</i> The checkpoint region points to the last segment written and the location of the inode map. <i>Id.</i>
19. An article of manufacture as in claim 18, wherein the instructions cause the processor to move atomically said file system on said storage system to said second consistent state by flushing said changes from said buffers to said storage system before updating said on-disk root inode with information from said incore root inode.	Rosenblum and Ousterhout teaches a checkpoint, which marks a consistent state of the file system. Rosenblum and Ousterhout, Sec. 3, p. 3, Sec. 4.1, p. 9. Once all modifications are written to disk, a checkpoint region is written, atomically moving the file system to a new consistent state. <i>Id.</i> The checkpoint region points to the last segment written and the location of the inode map. <i>Id.</i>
20. An article of manufacture as in claim 19, wherein updating said on-disk root inode comprises updating said on-disk root inode and then a copy of said on-disk root inode such that if updating said on-disk root inode is interrupted, said copy of said on-disk root inode still points to said first consistent state of said file system.	Replicating the metadata, such as a root inode, is a technique that is well known in the art. For instance, Leffler teaches that in the Unix filesystem the superblock is replicated to allow recovery from crashes that corrupt the primary copy of the superblock. Leffler, Sec. 7.3, p. 196. Furthermore, Rosenblum and Ousterhout teaches that the checkpoint region is replicated: "In order to handle a crash during a checkpoint operation there are actually two checkpoint regions, and checkpoint operation alternates between them." Rosenblum and Ousterhout, Sec. 4.1, p. 9. In view of the knowledge in the art and the teachings of Leffler and Rosenblum and Ousterhout, it would have

	been obvious for one of ordinary skill to make a copy of the root inode to recover from crashes that corrupt the primary copy of the root inode.
21. An article of manufacture as in claim 17, wherein the instructions further cause the processor to create a snapshot of said file system by copying only said on-disk root inode.	Rosenblum and Ousterhout teaches a checkpoint, which marks a consistent state of the file system. Rosenblum and Ousterhout, Sec. 3, p. 3, Sec. 4.1, p. 9. Once all modifications are written to disk, a checkpoint region is written, atomically moving the file system to a new consistent state. <i>Id.</i> The checkpoint region points to the last segment written and the location of the inode map. <i>Id.</i> The checkpoint represents a snapshot of the file system state at the time of the checkpoint.
22. An article of manufacture as in claim 21, wherein the instructions cause the processor to create said snapshot so that said snapshot and said file system share said first set of blocks on said storage system when said snapshot is created.	Rosenblum and Ousterhout teaches a checkpoint, which marks a consistent state of the file system. Rosenblum and Ousterhout, Sec. 3, p. 3, Sec. 4.1, p. 9. Once all modifications are written to disk, a checkpoint region is written, atomically moving the file system to a new consistent state. <i>Id.</i> The checkpoint region points to the last segment written and the location of the inode map. <i>Id.</i> The checkpoint represents a snapshot of the file system state at the time of the checkpoint. The LFS file system does not overwrite old data blocks, instead data is written out in a log format. <i>See e.g.</i> Rosenblum and Ousterhout, Fig. 1 (left side showing LFS). Unmodified data blocks therefore are shared between the checkpoint and the live file system. <i>Id.</i>

### Ninth Basis of Invalidity

The reference applicable to the ninth basis of invalidity is U.S. Patent 5,379,291 ("Belsan et. al").

The pertinence and manner of applying Belsan et al. to claims 1, 9, and 17 for which re-examination is requested is as follows:

<b>Claims of '211 Patent</b>	<b>Belsan</b>
1. A method of maintaining a file system stored in a memory and on a storage system that includes one or more hard disks, said method	Belsan teaches a method of maintaining a file system stored in a cache memory and on a disk storage system. <i>See Fig.1.</i>

comprising steps of:	
maintaining an on-disk root inode on said storage system, said on-disk root inode pointing directly and indirectly to a first set of blocks on said storage system that store a first consistent state of said file system; and	Belsan teaches a mapping table and a copy table. The mapping table (root inode) is maintained on disk and contains pointers to data blocks stored on disk. Col. 7:46-66. The mapping table is stored (backed up) on disk. Col. 6:36-39. Mapping tables store a plurality of pointers identifying data blocks. Col. 8:64-9:27, Figs. 2 and 3. The disclosed mapping table points to blocks that store a consistent state.
maintaining an incore root inode in said memory, said incore root inode pointing directly and indirectly to buffers in said memory and a second set of blocks on said storage system, said buffers and said second set of blocks storing data and metadata for a second consistent state of said file system, said second set of blocks including at least some blocks in said first set of blocks, with changes between said first consistent state and said second consistent state being stored in said buffers and in ones of said second set of blocks not pointed to by said on-disk inode.	Belsan further teaches that the mapping table is uploaded and modified in cache memory during write transactions, constituting an incore root inode. Contents of the on-disk mapping table are loaded into a cache hash table. <i>See e.g.</i> Col. 13:54-59. The disclosed file system uses copy-on-write, so that a block to be written is first copied into cache memory and then is re-written to a new location on disk. Col. 12:22-38; 13:42-14:6. During a sequence of write operations, the cached mapping table and associated copy table point to some data buffers in cache (buffers that are currently being written) as well as some modified blocks. The cached mapping table also points to unmodified blocks on disk, which are also pointed to by the un-updated mapping table stored on-disk.
9. A device comprising: a processor; a memory; and	Belsan Fig. 1 discloses that the file system runs on a processor 101 and includes a cache memory 113. More specifically, Fig. 4 shows processor 204-0 and cache memory 113. Col. 6:2-27.
a storage system including one or more hard disks;	Belsan Fig. 1 discloses a storage system including one or more hard disks 103-1 (disk drive subset).
wherein said memory and said storage system store a file system; and	Data storage subsystem 100 is a file system. Col. 3:34-36.
wherein said memory also stores information including instructions executable by said processor to maintain said file system, the instructions including steps of (a) maintaining an on-disk root inode on said storage system, said on-disk root inode pointing directly and indirectly to a first set of blocks on said storage system that store a first consistent state of said file system, and (b) maintaining an incore root inode in	<p>Belsan teaches a mapping table and a copy table. The mapping table (root inode) is maintained on disk and contains pointers to data blocks stored on disk. Col. 7:46-66. The mapping table is stored (backed up) on disk. Col. 6:36-39. Mapping tables store a plurality of pointers identifying data blocks. Col. 8:64-9:27, Figs. 2 and 3. The disclosed mapping table points to blocks that store a consistent state.</p> <p>Belsan further teaches that the mapping table is uploaded and modified in cache memory during write transactions, constituting an incore root inode. Contents</p>

<p>said memory, said incore root inode pointing directly and indirectly to buffers in said memory and a second set of blocks on said storage system, said buffers and said second set of blocks storing data and metadata for a second consistent state of said file system, said second set of blocks including at least some blocks in said first set of blocks, with changes between said first consistent state and said second consistent state being stored in said buffers and in ones of said second set of blocks not pointed to by said on-disk inode.</p>	<p>of the on-disk mapping table are loaded into a cache hash table. <i>See e.g.</i> Col. 13:54-59. The disclosed file system uses copy-on-write, so that a block to be written is first copied into cache memory and then is re-written to a new location on disk. Col. 12:22-38; 13:42-14:6. During a sequence of write operations, the cached mapping table and associated copy table point to some data buffers in cache (buffers that are currently being written) as well as some modified blocks. The cached mapping table also points to unmodified blocks on disk, which are also pointed to by the un-updated mapping table stored on-disk.</p>
<p>17. An article of manufacture comprising a machine-readable storage medium storing instructions executable by a processor coupled to a memory and to a storage system, said storage system comprising one or more hard disks, said memory and said storage system storing a file system, wherein the instructions, when executed by the processor, cause the processor to (a) maintain an on-disk root inode on said storage system, said on-disk root inode pointing directly and indirectly to a first set of blocks on said storage system that store a first consistent state of said file system, and (b) maintain an incore root inode in said memory, said incore root inode pointing directly and indirectly to buffers in said memory and a second set of blocks on said storage system, said buffers and said second set of blocks storing data and metadata for a second consistent state of said file system, said second set of blocks including at least some blocks in said first set of blocks, with changes between said first consistent state and said second consistent state being</p>	<p>Belsan Fig. 1 discloses that the file system runs on a processor 101 and includes a cache memory 113. More specifically, Fig. 4 shows processor 204-0 and cache memory 113. Col. 6:2-27. Instructions to run the Belsan file system are inherently stored on machine-readable medium. Belsan Fig. 1 discloses a storage system including one or more hard disks 103-1 (disk drive subset).</p> <p>Belsan teaches a mapping table and a copy table. The mapping table (root inode) is maintained on disk and contains pointers to data blocks stored on disk. Col. 7:46-66. The mapping table is stored (backed up) on disk. Col. 6:36-39. Mapping tables store a plurality of pointers identifying data blocks. Col. 8:64-9:27, Figs. 2 and 3. The disclosed mapping table points to blocks that store a consistent state.</p> <p>Belsan further teaches that the mapping table is uploaded and modified in cache memory during write transactions, constituting an incore root inode. Contents of the on-disk mapping table are loaded into a cache hash table. <i>See e.g.</i> Col. 13:54-59. The disclosed file system uses copy-on-write, so that a block to be written is first copied into cache memory and then is re-written to a new location on disk. Col. 12:22-38; 13:42-14:6. During a sequence of write operations, the cached mapping table and associated copy table point to some data buffers in cache (buffers that are currently being written) as well as some modified blocks. The cached</p>

stored in said buffers and in ones of said second set of blocks not pointed to by said on-disk inode.	mapping table also points to unmodified blocks on disk, which are also pointed to by the un-updated mapping table stored on-disk.
---	---

### Tenth Basis of Invalidity

The reference applicable to the tenth basis of invalidity is U.S. Patent 5,218,695 ("Noveck").

The pertinence and manner of applying Noveck. to claims 1, 9, and 17 for which re-examination is requested is as follows:

<b>Claims of '211 Patent</b>	<b>Noveck</b>
1. A method of maintaining a file system stored in a memory and on a storage system that includes one or more hard disks, said method comprising steps of:	Noveck teaches a server including one or more hard disks and maintaining a file system. <i>See</i> Fig.1, Col. 6:20-24.
maintaining an on-disk root inode on said storage system, said on-disk root inode pointing directly and indirectly to a first set of blocks on said storage system that store a first consistent state of said file system; and	In particular, Noveck discloses an on-disk Unix inode, with direct and indirect blocks. Col. 3:29-35, 3:60-63.
maintaining an incore root inode in said memory, said incore root inode pointing directly and indirectly to buffers in said memory and a second set of blocks on said storage system, said buffers and said second set of blocks storing data and metadata for a second consistent state of said file system, said second set of blocks including at least some blocks in said first set of blocks, with changes between said first consistent state and said second consistent state being stored in said buffers and in ones of said second set of blocks not pointed to by said on-disk inode.	Noveck further teaches an in-core inode, which is an image of the on-disk inode, maintained in cache memory. Col. 6:25-31. As data in the file system is modified, the in-core structure differs from the on-disk inode and data. Col. 7: 14-24. The in-core inode points to blocks and buffers that have been modified since the on-disk inode was written.
9. A device comprising: a processor; a	Noveck teaches a server comprising a processor and a

memory; and	memory. <i>See</i> Fig. 1, Col. 7:14-17.
a storage system including one or more hard disks;	Noveck teaches a server including one or more hard disks and maintaining a file system. <i>See</i> Fig.1, Col. 6:20-24.
wherein said memory and said storage system store a file system; and	Noveck teaches a server including one or more hard disks, a cache memory and maintaining a file system. <i>See</i> Fig.1, Col. 6:20-24, Col. 7:14-17.
wherein said memory also stores information including instructions executable by said processor to maintain said file system, the instructions including steps of (a) maintaining an on-disk root inode on said storage system, said on-disk root inode pointing directly and indirectly to a first set of blocks on said storage system that store a first consistent state of said file system, and (b) maintaining an incore root inode in said memory, said incore root inode pointing directly and indirectly to buffers in said memory and a second set of blocks on said storage system, said buffers and said second set of blocks storing data and metadata for a second consistent state of said file system, said second set of blocks including at least some blocks in said first set of blocks, with changes between said first consistent state and said second consistent state being stored in said buffers and in ones of said second set of blocks not pointed to by said on-disk inode.	<p>Noveck discloses an on-disk Unix inode, with direct and indirect blocks. Col. 3:29-35, 3:60-63.</p> <p>Noveck further teaches an in-core inode, which is an image of the on-disk inode, maintained in cache memory. Col. 6:25-31. As data in the file system is modified, the in-core structure differs from the on-disk inode and data. Col. 7: 14-24. The in-core inode points to blocks and buffers that have been modified since the on-disk inode was written.</p>
17. An article of manufacture comprising a machine-readable storage medium storing instructions executable by a processor coupled to a memory and to a storage system, said storage system comprising one or more hard disks, said memory and said storage system storing a file system, wherein the instructions, when executed by the processor, cause the processor to (a) maintain an	<p>Noveck teaches a server including one or more hard disks, a cache memory and maintaining a file system. <i>See</i> Fig.1, Col. 6:20-24, Col. 7:14-17. Instructions for maintaining the Noveck file system are inherently stored on a machine-readable medium.</p> <p>Noveck discloses an on-disk Unix inode, with direct and indirect blocks. Col. 3:29-35, 3:60-63.</p> <p>Noveck further teaches an in-core inode, which is an image of the on-disk inode, maintained in cache</p>

on-disk root inode on said storage system, said on-disk root inode pointing directly and indirectly to a first set of blocks on said storage system that store a first consistent state of said file system, and (b) maintain an incore root inode in said memory, said incore root inode pointing directly and indirectly to buffers in said memory and a second set of blocks on said storage system, said buffers and said second set of blocks storing data and metadata for a second consistent state of said file system, said second set of blocks including at least some blocks in said first set of blocks, with changes between said first consistent state and said second consistent state being stored in said buffers and in ones of said second set of blocks not pointed to by said on-disk inode.	memory. Col. 6:25-31. As data in the file system is modified, the in-core structure differs from the on-disk inode and data. Col. 7: 14-24. The in-core inode points to blocks and buffers that have been modified since the on-disk inode was written.
---	---

#### Eleventh Basis of Invalidity

The reference applicable to the eleventh basis of invalidity is Gray et al., *The Recovery Manager of the System R Database Manager*, ACM, 1981 ("Gray").

The pertinence and manner of applying Gray. to claims 1-6, 9-14, 17-22 for which re-examination is requested is as follows:

<b>Claims of '211 Patent</b>	<b>Gray</b>
1. A method of maintaining a file system stored in a memory and on a storage system that includes one or more hard disks, said method comprising steps of:	Gray teaches a database system using shadowing (copy-on-write) to manage files. <i>See e.g.</i> Gray, Sec. 2.1, Fig. 7. The database in Gray inherently resides in memory and on disk.
maintaining an on-disk root inode on said storage system, said on-disk root inode pointing directly and indirectly to a first set of blocks on said storage system that store a first consistent state of said file system; and	A copy of a page table (equivalent to an inode) and data pages (equivalent to data blocks) as well as a directory root (equivalent to root inode) is cached in memory and resident on disk. Gray, Sec. 2.1, Sec. 2.7, Fig. 10.
maintaining an incore root inode in	A copy of a page table (equivalent to an inode) and data

<p>said memory, said incore root inode pointing directly and indirectly to buffers in said memory and a second set of blocks on said storage system, said buffers and said second set of blocks storing data and metadata for a second consistent state of said file system, said second set of blocks including at least some blocks in said first set of blocks, with changes between said first consistent state and said second consistent state being stored in said buffers and in ones of said second set of blocks not pointed to by said on-disk inode.</p>	<p>pages (equivalent to data blocks) as well as a directory root (equivalent to root inode) is cached in memory and resident on disk. Gray, Sec. 2.1, Sec. 2.7, Fig. 10. Gray teaches that the database takes periodic checkpoints, representing consistent system states by flushing the cache to disk and then writing to disk an updated directory root. Gray, Sec. 2.7, Fig. 10. It is inherent that the in-memory copy of the directory root points to buffers and blocks representing the database in a present consistent state. Changes between the present consistent state and the checkpointed previous state are inherently stored in buffers and blocks.</p>
<p>2. A method as in claim 1, wherein said file system on said storage system always moves atomically from said first consistent state to said second consistent state.</p>	<p>Gray teaches that the database takes periodic checkpoints, representing consistent system states by flushing the cache to disk and then writing to disk an updated directory root. Gray, Sec. 2.7, Fig. 10.</p>
<p>3. A method as in claim 2, wherein said file system on said storage system moves atomically to said second consistent state by flushing said changes from said buffers to said storage system before updating said on-disk root inode with information from said incore root inode.</p>	<p>Gray teaches that the database takes periodic checkpoints, representing consistent system states by flushing the cache to disk and then writing to disk an updated directory root. Gray, Sec. 2.7, Fig. 10.</p>
<p>4. A method as in claim 3, wherein updating said on-disk root inode further comprises updating said on-disk root inode and then a copy of said on-disk root inode such that if updating said on-disk root inode is interrupted, said copy of said on-disk root inode still points to said first consistent state of said file system.</p>	<p>Gray teaches that the "directory root is duplexed on disk to tolerate failures while writing the directory root." Gray, Sec. 2.7.</p>
<p>5. A method as in claim 1, further comprising the step of creating a snapshot of said file system by copying only said on-disk root inode.</p>	<p>Gray teaches that the database takes periodic checkpoints, representing consistent system states by flushing the cache to disk and then writing to disk an updated directory root. Gray, Sec. 2.7, Fig. 10.</p>

6. A method as in claim 5, wherein when said snapshot is created, said snapshot and said file system share said first set of blocks on said storage system.	Gray teaches that the database takes periodic checkpoints, representing consistent system states by flushing the cache to disk and then writing to disk an updated directory root. Gray, Sec. 2.7, Fig. 10. The checkpoint and the current database state inherently share a set of blocks that have not changed between the checkpoint and the current database state.
9. A device comprising: a processor; a memory; and	Gray database inherently operates on a computing device having a processor and a memory.
a storage system including one or more hard disks;	Gray database inherently resides on one or more hard disks.
wherein said memory and said storage system store a file system; and	Gray teaches a database system using shadowing (copy-on-write) to manage files. <i>See e.g.</i> Gray, Sec. 2.1, Fig. 7. The database in Gray inherently resides in memory and on disk.
wherein said memory also stores information including instructions executable by said processor to maintain said file system, the instructions including steps of (a) maintaining an on-disk root inode on said storage system, said on-disk root inode pointing directly and indirectly to a first set of blocks on said storage system that store a first consistent state of said file system, and (b) maintaining an incore root inode in said memory, said incore root inode pointing directly and indirectly to buffers in said memory and a second set of blocks on said storage system, said buffers and said second set of blocks storing data and metadata for a second consistent state of said file system, said second set of blocks including at least some blocks in said first set of blocks, with changes between said first consistent state and said second consistent state being stored in said buffers and in ones of said second set of blocks not pointed to by said on-disk inode.	A copy of a page table (equivalent to an inode) and data pages (equivalent to data blocks) as well as a directory root (equivalent to root inode) is cached in memory and resident on disk. Gray, Sec. 2.1, Sec. 2.7, Fig. 10. Gray teaches that the database takes periodic checkpoints, representing consistent system states by flushing the cache to disk and then writing to disk an updated directory root. Gray, Sec. 2.7, Fig. 10. It is inherent that the in-memory copy of the directory root points to buffers and blocks representing the database in a present consistent state. Changes between the present consistent state and the checkpointed previous state are inherently stored in buffers and blocks.
10. A device as in claim 9, wherein said file system on said storage	Gray teaches that the database takes periodic checkpoints, representing consistent system states by

system always moves atomically from said first consistent state to said second consistent state.	flushing the cache to disk and then writing to disk an updated directory root. Gray, Sec. 2.7, Fig. 10.
11. A device as in claim 10, wherein said file system on said storage system moves atomically to said second consistent state by flushing said changes from said buffers to said storage system before updating said on-disk root inode with information from said incore root inode.	Gray teaches that the database takes periodic checkpoints, representing consistent system states by flushing the cache to disk and then writing to disk an updated directory root. Gray, Sec. 2.7, Fig. 10.
12. A device as in claim 11, wherein updating said on-disk root inode further comprises updating said on-disk root inode and then a copy of said on-disk root inode such that if updating said on-disk root inode is interrupted, said copy of said on-disk root inode still points to said first consistent state of said file system.	Gray teaches that the “directory root is duplexed on disk to tolerate failures while writing the directory root.” Gray, Sec. 2.7.
13. A device as in claim 9, wherein the instructions further comprise the step of creating a snapshot of said file system by copying only said on-disk root inode.	Gray teaches that the database takes periodic checkpoints, representing consistent system states by flushing the cache to disk and then writing to disk an updated directory root. Gray, Sec. 2.7, Fig. 10.
14. A device as in claim 13, wherein when said snapshot is created, said snapshot and said file system share said first set of blocks on said storage system.	Gray teaches that the database takes periodic checkpoints, representing consistent system states by flushing the cache to disk and then writing to disk an updated directory root. Gray, Sec. 2.7, Fig. 10. The checkpoint and the current database state inherently share a set of blocks that have not changed between the checkpoint and the current database state.
17. An article of manufacture comprising a machine-readable storage medium storing instructions executable by a processor coupled to a memory and to a storage system, said storage system comprising one or more hard disks, said memory and said storage system storing a file system, wherein the instructions,	Gray teaches a database system using shadowing (copy-on-write) to manage files. <i>See e.g.</i> Gray, Sec. 2.1, Fig. 7. The database in Gray inherently resides in memory and on disk.  A copy of a page table (equivalent to an inode) and data pages (equivalent to data blocks) as well as a directory root (equivalent to root inode) is cached in memory and resident on disk. Gray, Sec. 2.1, Sec. 2.7, Fig. 10. Gray

when executed by the processor, cause the processor to (a) maintain an on-disk root inode on said storage system, said on-disk root inode pointing directly and indirectly to a first set of blocks on said storage system that store a first consistent state of said file system, and (b) maintain an incore root inode in said memory, said incore root inode pointing directly and indirectly to buffers in said memory and a second set of blocks on said storage system, said buffers and said second set of blocks storing data and metadata for a second consistent state of said file system, said second set of blocks including at least some blocks in said first set of blocks, with changes between said first consistent state and said second consistent state being stored in said buffers and in ones of said second set of blocks not pointed to by said on-disk inode.	teaches that the database takes periodic checkpoints, representing consistent system states by flushing the cache to disk and then writing to disk an updated directory root. Gray, Sec. 2.7, Fig. 10. It is inherent that the in-memory copy of the directory root points to buffers and blocks representing the database in a present consistent state. Changes between the present consistent state and the checkpointed previous state are inherently stored in buffers and blocks.
18. An article of manufacture as in claim 17, wherein the instructions further cause the processor to move atomically said file system on said storage system from said first consistent state to said second consistent state.	Gray teaches that the database takes periodic checkpoints, representing consistent system states by flushing the cache to disk and then writing to disk an updated directory root. Gray, Sec. 2.7, Fig. 10.
19. An article of manufacture as in claim 18, wherein the instructions cause the processor to move atomically said file system on said storage system to said second consistent state by flushing said changes from said buffers to said storage system before updating said on-disk root inode with information from said incore root inode.	Gray teaches that the database takes periodic checkpoints, representing consistent system states by flushing the cache to disk and then writing to disk an updated directory root. Gray, Sec. 2.7, Fig. 10.
20. An article of manufacture as in claim 19, wherein updating said on-	Gray teaches that the "directory root is duplexed on disk to tolerate failures while writing the directory root."

disk root inode comprises updating said on-disk root inode and then a copy of said on-disk root inode such that if updating said on-disk root inode is interrupted, said copy of said on-disk root inode still points to said first consistent state of said file system.	Gray, Sec. 2.7.
21. An article of manufacture as in claim 17, wherein the instructions further cause the processor to create a snapshot of said file system by copying only said on-disk root inode.	Gray teaches that the database takes periodic checkpoints, representing consistent system states by flushing the cache to disk and then writing to disk an updated directory root. Gray, Sec. 2.7, Fig. 10.
22. An article of manufacture as in claim 21, wherein the instructions cause the processor to create said snapshot so that said snapshot and said file system share said first set of blocks on said storage system when said snapshot is created.	Gray teaches that the database takes periodic checkpoints, representing consistent system states by flushing the cache to disk and then writing to disk an updated directory root. Gray, Sec. 2.7, Fig. 10. The checkpoint and the current database state inherently share a set of blocks that have not changed between the checkpoint and the current database state.

### **III. STATEMENT POINTING OUT SUBSTANTIAL NEW QUESTION OF PATENTABILITY**

Since claims 1-24 of the '211 Patent are not patentable over the prior art references cited above for the reasons set forth above, a substantial new question of patentability is raised for each claim. Further, these prior art references cited above are material to the subject matter of the '211 Patent. In particular, these prior art references provide teachings not provided during the prosecution of the '211 Patent. Therefore, a substantial new question of patentability has been raised, and reexamination is respectfully requested.

### **CONCLUSION**

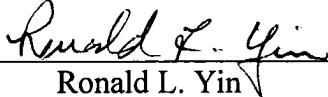
Based on the above remarks, it is respectfully submitted that a substantial new question of patentability has been raised with respect to Claims 1-24 of the '211 Patent. Therefore, reexamination of Claims 1-24 is respectfully requested.

Any fee due for this reexamination may be charged to Deposit Account No. 07-1896.

Respectfully submitted,

**DLA PIPER US LLP**

Date: January 14, 2008

By:   
Ronald L. Yin  
Reg. No. 27,607

Attorneys for Applicant(s)

Ronald L. Yin  
**DLA Piper** US LLP  
2000 University Avenue  
East Palo Alto, CA 94303-2248  
650-833-2437 (Direct)  
650-833-2000 (Main)  
650-833-2001 (Facsimile)  
ronald.yin@dlapiper.com

1 MARK D. FOWLER, Bar No. 124235  
mark.fowler@dlapiper.com  
2 DAVID ALBERTI, Bar No. 220625  
david.alberti@dlapiper.com  
3 CHRISTINE K. CORBETT, Bar No. 209128  
christine.corbett@dlapiper.com  
4 YAKOV M. ZOLOTOREV, Bar No. 224260  
yakov.zolotorev@dlapiper.com  
5 CARRIE L. WILLIAMSON, Bar No. 230873  
carrie.williamson@dlapiper.com

6 DLA PIPER US LLP  
7 2000 University Avenue  
East Palo Alto, CA 94303-2214  
8 Tel: 650.833.2000  
Fax: 650.833.2001

9 Attorneys for Defendant/Counterclaimant,  
10 Sun Microsystems, Inc.

11 UNITED STATES DISTRICT COURT  
12 NORTHERN DISTRICT OF CALIFORNIA  
13 SAN FRANCISCO DIVISION  
14

15 NETWORK APPLIANCE, INC.

16 Plaintiff-Counterclaim Defendant,

17 v.

18 SUN MICROSYSTEMS, INC., a Delaware  
corporation,,

19 Defendant-Counterclaimant.  
20

CASE NO. C-07-06053 EDL

**PROOF OF SERVICE**

21 I am a resident of the State of California, over the age of eighteen years, and not a party to  
22 the within action. My business address is DLA Piper US LLP, 2000 University Avenue, East  
Palo Alto, California 94303-2214. On January 14, 2008, I served the within documents:

- 23
- 24 • **NOTICE OF FAILURE TO COMPLY WITH *INTER PARTES***  
**REEXAMINATION REQUEST FILING REQUIREMENTS**
  - 25 • **RESPONSE TO NOTICE OF FAILURE TO COMPLY WITH *INTER***  
***PARTES* REEXAMINATION REQUEST FILING REQUIREMENTS**
  - 26 • **REPLACEMENT ATTACHMENT TO REQUEST FOR INTER-PARTES**  
27 **RE-EXAMINATION OF U.S. PATENT NO. 6,892,211**
- 28

- 1 ☐ by transmitting via facsimile the document(s) listed above to the fax number(s) set  
2 forth below on this date before 5:00 p.m.
- 3 ☐ by placing the document(s) listed above in a sealed envelope with postage thereon  
4 fully prepaid, in the United States mail at East Palo Alto, California addressed as  
5 set forth below.
- 6 ☐ by personally delivering the document(s) listed above to the person(s) at the  
7 address(es) set forth below.
- 8 ☒ by transmitting via the internet the document(s) listed above to the e-mail  
9 addressee(s) as set forth below.

10 **Attorney for Network Appliance, Inc.**

11 Jeffrey G. Homrig, Esq.  
12 Weil Gotshal & Manges  
13 201 Redwood Shores Parkway  
14 Redwood Shores, CA 94065  
15 Tel: (650) 802-3000  
16 Fax: (650) 802-3100  
17 Email: [jeffrey.homrig@weil.com](mailto:jeffrey.homrig@weil.com)

18 I am readily familiar with the firm's practice of collection and processing correspondence  
19 for mailing. Under that practice it would be deposited with the U.S. Postal Service on that same  
20 day with postage thereon fully prepaid in the ordinary course of business. I am aware that on  
21 motion of the party served, service is presumed invalid if postal cancellation date or postage  
22 meter date is more than one day after date of deposit for mailing in affidavit.

23 I declare that I am employed in the office of a member of the Bar of or permitted to  
24 practice before this Court at whose direction the service was made.

25 Executed on January 14, 2008, at East Palo Alto, California.

26 

27 Carrie L. Williamson